Documentation interne

Introduction

Ce document est la documentation interne détaillant chaque module composant l'application de GED automatique. L'application de GED automatique est une application web dont le côté client a été réalisé en React et Scss et le côté serveur en Node.js, Express et avec l'API d'OpenAI.

Frontend

Cette section décrit les modules du côté client. Les "assets" incluant les images, logos et polices d'écriture ne seront pas mentionnés.

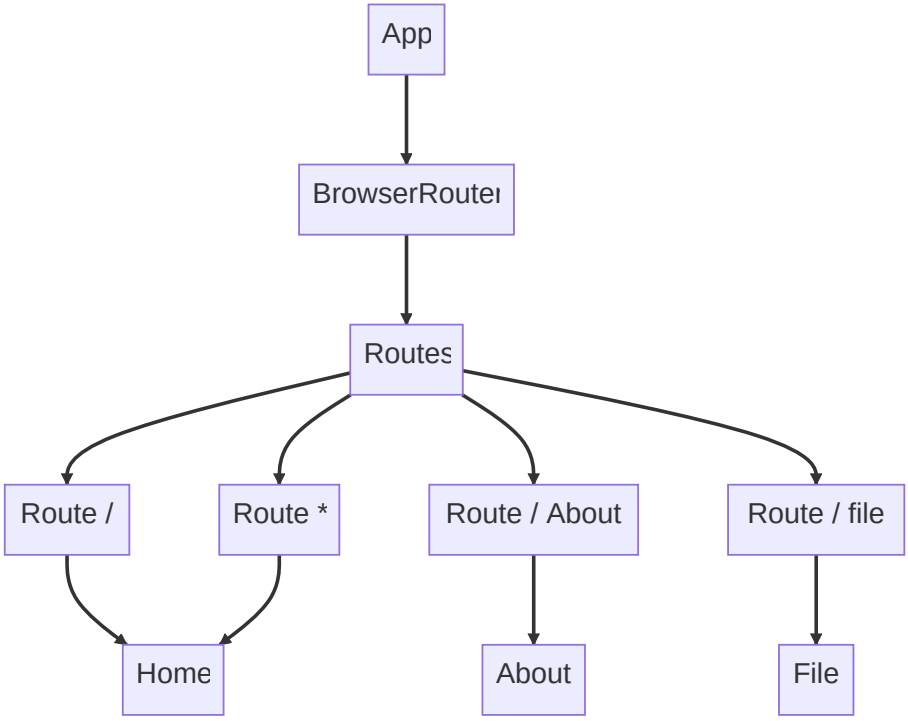
Conteneurs

Le côté client est structuré suivant le patron de conception "conteneur" couramment appelé dans la communauté React "composant bête" / "composant intelligent". Un composant bête en React, aussi appelé composant de présentation car sa seule responsabilité est de présenter quelque chose, se contente seulement de retourner des éléments de l'interface utilisateur et ne peut pas modifier ces données. Un composant "intelligent" gère les états de l'application, interagit avec les API et le serveur, peut contenir d'autres composants (enfants) et leur passer des données, d'où l'appellation de conteneurs. Les conteneurs utilisés dans l'application de GED sont :

App.jsx

- **Objectifs**

`function App()` est le composant qui définit les routes et les composants correspondant à différents chemins dans l'application. Il gère l'état du système de routage et affiche les composants enfants appropriés selon le chemin de l'URL courante.



De plus, ce composant utilise le routeur de React `BrowserRouter` et les composants `Routes` et `Route` pour gérer la logique de routage.

- **Fonctions**

Le composant `App` ne contient pas d'autres fonctions.

- **Attributs et types**

Le composant `App` ne contient pas d'attributs en particulier.

- **Relations d'utilisation**

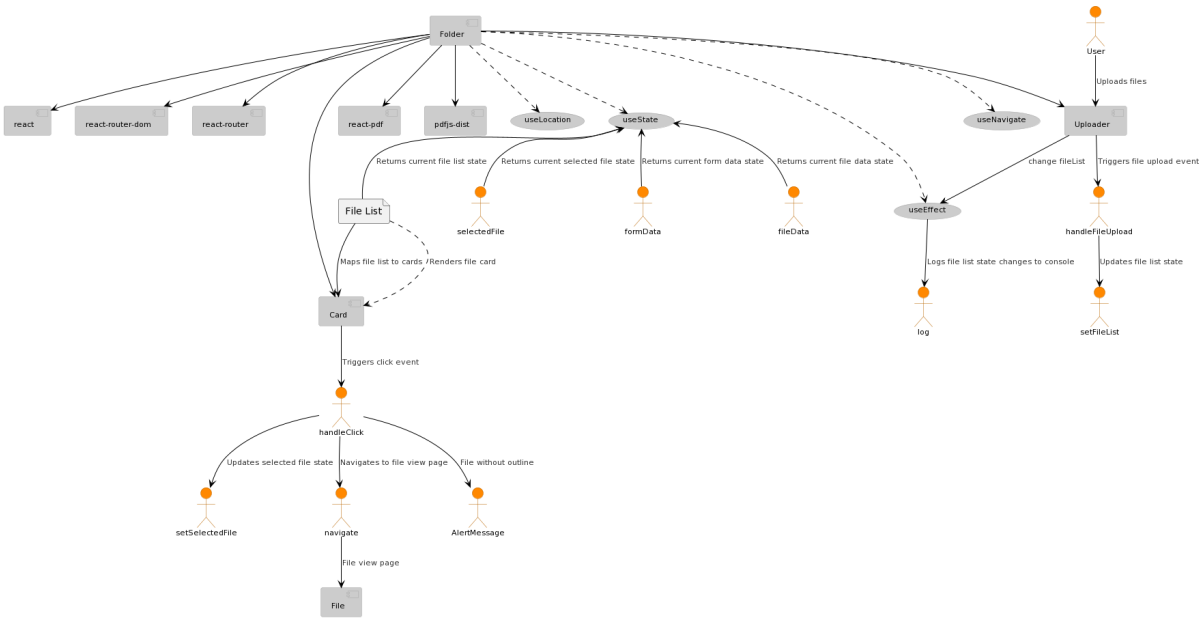
Le composant App est le composant racine. Il s'agit du composant de plus haut niveau qui contient tous les autres composants de l'application et permet d'initialiser l'etat global de l'application ainsi que définir les routes. Il utilise donc indirectement tous les composants mais n'est utilisé par aucun autre composant.

Folder.jsx

- **Objectifs**

`const Folder` permet à l'utilisateur de choisir le fichier qu'il souhaite traiter. Plus précisément, avec le composant `Folder`, l'utilisateur peut parcourir des fichiers PDF, affiche les fichiers téléchargés sous forme de liste de cartes et naviguer vers une page de visualisation de fichier lorsqu'une carte est cliquée si le fichier contient bien une table de matières, sinon affiche une notification popup d'erreur à l'utilisateur.

Le composant utilise des hooks React tels que useState, useEffect, et useLocation pour gérer son état et effectuer des tâches asynchrones. Il importe également un autre composant Card qui est utilisé pour afficher chaque fichier téléchargé dans une carte.



- **Fonctions**

```
async function handleFileUpload(event)
```

- Description
  - Cette fonction lit le contenu des fichiers sélectionnés, extrait les miniatures des fichiers PDF (si nécessaire) et ajoute les fichiers à la liste des fichiers téléchargés. La liste des fichiers est mise à jour en utilisant le hook useState.
- Paramètres
  - event : L'évènement onChange sur l'élément HTML `<input type="file" onChange={handleFileUpload}` correspondant à l'action de parcourir des fichiers et permet de récupérer les données de ces fichiers

```
const files = event.target.files;
```

- Retourne
  - Rien, elle modifie directement l'état de l'attribut fileList

```
setFileList(fileList => [...fileList, { name: file.name, thumbnail, fileObject: file }]);
```

```
function closeAlert()
```

- Description
  - Cette fonction ferme le message d'alerte affiché lorsque l'utilisateur clique sur la croix du message d’alert après avoir déclanché l’alerte en cliquant sur un fichier sans table de matières.
- Paramètres
  - Aucun.
- Retourne
  - Rien. Elle met simplement à jour la classe CSS de l'élément HTML correspondant au message d'alerte pour le cacher.

```
alert.classList.add("hide");
```

```
function handleClick(index)
```

- Description
  - Cette fonction est appelée quand l'utilisateur clique sur une carte de fichier. Elle met à jour l'état du fichier sélectionné et extrait les articles du PDF, puis navigue vers la page de vue de fichier.
- Paramètres
  - index : le numéro d'index du fichier sélectionné dans la liste des fichiers téléchargé

```
setSelectedFile(fileList[index]);
const file = fileList[index].fileObject;
```

- Retourne
  - Rien. Elle traite le fichier en appelant les autres fonctions du composant (voir ci dessous) et redirige l’utilisateur vers la page “/file” affichant la gestion du fichier cliqué.

    Le premièr argument passé à `navigate` est l'URL de la nouvelle page vers laquelle on veut naviguer, dans ce cas-ci `'/file'`.

    Le deuxième argument est un objet JavaScript nommé `state` qui contient des informations liées à la nouvelle page. L'objet `state` est enregistré dans l'historique de navigation du navigateur et peut être récupéré lorsque l'utilisateur retourne sur la page précédente.

```
navigate('/file', {
                state: {
                    fileList: fileList,
                    selectedFile: fileList[index],
                    fileName: fileList[index].name,
                    fileThumbnail: thumbnail,
                    fileContents: fileContents,
                    articleData: articleData
                }
            });
```

function extractArticles(outline,fileContents)

- Description
  - Cette fonction extrait les articles d'un fichier PDF en utilisant la table des matières et en essayant d'identifier chaque header / titre de la table de matières dans le contenu du fichier PDF. Elle utilise la manipulation de chaînes pour trouver les positions de début et de fin de chaque article dans le paramètre `fileContents`.

    ```
    startIndex = fileContentsNoSpaces.indexOf(titleNoSpaces, startIndex + titleNoSpaces.length);
    ```

    Il crée ensuite un objet appelé `articleData` qui contient les données extraites pour chaque article.

    ```
    articleData[i] = { title: outline[i].title, content: fileContentsNoSpaces.slice(start, end).trim() };
    ```

- Paramètres
  - outline : Tableau d'objets contenant la table de matières du fichier
  - fileContents : Chaine de caractères contenant le texte du fichier
- Retourne
  - La fonction retourne un objet `articleData` contenant les titres et le contenu de chaque article dans le fichier PDF.

function readPdfFile(file)

- Description

- Cette fonction lit un fichier PDF et renvoie son image miniature, le nombre de pages, le texte extrait et la table des matières (le cas échéant).

```
const reader = new FileReader();
const fileDataPromise = new Promise(resolve => {
    reader.onload = () => resolve(reader.result);
});
reader.readAsArrayBuffer(file);
```

- Paramètres
  - file: Objet File correspondant au fichier PDF à traiter
- Retourne
  - La fonction retourne un objet contenant l'image miniature, le nombre de pages, le texte extrait et la table des matières (le cas échéant) du fichier PDF.

```
return { thumbnail, pageCount, text, outline };
```

- **Attributs et types**

Le composant `Folder` utilise les hooks `useState`, `useEffect`, `useNavigate` et `useLocation` pour gérer son état et la navigation dans l'application. Il utilise le hook `useLocation` pour récupérer l'état actuel de l'URL, qui peut contenir des données de fichiers stockées dans le `state`. Il utilise également le hook `useNavigate` pour naviguer vers une autre page lorsque l'utilisateur clique sur une carte de fichier.

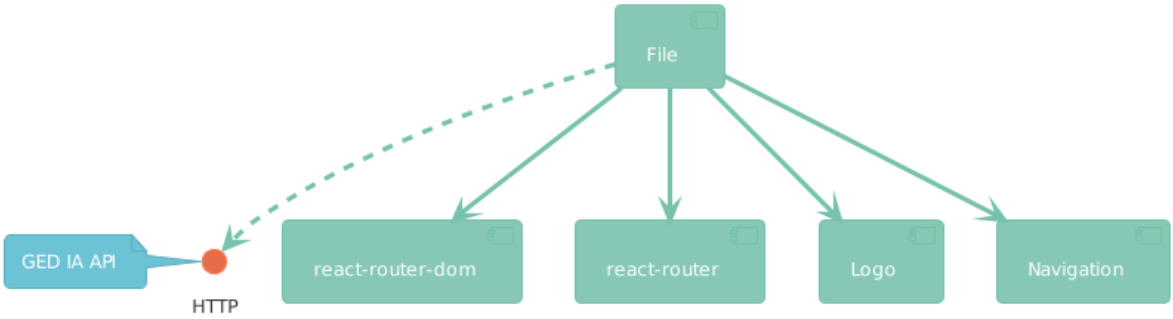
- **Relations d'utilisation**

Bien que ce composant ne reçoit pas de données et n'en transmet pas directement (avec props), il utilise le composant enfant `Card` et les composants importés depuis la bibliothèque `react-pdf` , `Document` et `Page` qui permettent de récupérer le contenu d'un fichier PDF dans l'application.

File.jsx

- **Objectifs**

Le composant `File` est une page qui affiche le contenu d'un fichier sélectionné et permet à l'utilisateur de générer un résumé, des mots clés et d'enregistrer ces derniers dans une base de données.



- **Fonctions**

function makeSummary()

- Description :
  - Cette fonction connecte le côté client et le côté serveur en envoyant au serveur les articles du fichier PDF, le champ prompt, le champ clé API et le nom du fichier.

```
const requestBody = {
            articleData: articleData,
            name: fileName,
            prompt: prompt,
            key: apiKey,
        };
```

Elle envoie une requête POST au serveur, extrait les données de la réponse, concatène les résumés générés et met à jour la zone de texte du résumé avec le résumé généré.

- Paramètres : Aucun
- Retour : Aucun

function openFile()

- Description :
  - Cette fonction ouvre le fichier dans un nouvel onglet lorsque la page de couverture est cliquée. Pour cela, elle utilise la méthode `open` de l'objet `window` et la méthode statique `createObjectURL` de l'interface `URL`.

```
window.open(URL.createObjectURL(file.fileObject), '_blank');
```

- Paramètres : Aucun
- Retour : Aucun

function saveDB()

- Description :
  - Cette fonction envoie au serveur les contenus des champs de saisie correspondant au résumé et aux mots clés ainsi que le nom du fichier.

```
const requestBody = {
            name: fileName,
            summary: summary,
            keywords: keyword
        };
```

Elle envoie une requête POST au serveur, affiche un message de succès ou un message d'erreur en fonction du statut de réponse du serveur.

- Paramètres : Aucun
- Retour : Aucun

function makeIndex()

- Description :
  - Cette fonction connecte le côté client et le côté serveur en envoyant au serveur les articles du fichier PDF, le champ prompt, le champ clé API et le nom du fichier.

```
const requestBody = {
            articleData: articleData,
            name: fileName,
            prompt: prompt,
            key: apiKey,
        };
```

Elle envoie une requête POST au serveur, extrait les données de la réponse, concatène les mots clés générés et met à jour le champ de mots clés avec les mots clés générés.

- Paramètres : Aucun
- Retour : Aucun

```
function closeAlert()
```

- Description :
  - Cette fonction ferme l'alerte pour une clé API invalide lorsque l’icone croix de l’alerte est cliquée.
- Paramètres : Aucun
- Retour : Aucun

```
function closeAlertBD()
```

- Description :
  - Cette fonction ferme l'alerte pour une sauvegarde réussie dans la base de données lorsque l’icone croix de l’alerte est cliquée.
- Paramètres : Aucun
- Retour : Aucun

```
function closeAlertEchecBD()
```

- Description :
  - Cette fonction ferme l'alerte pour une sauvegarde échouée dans la base de données lorsque l’icone croix de l’alerte est cliquée.

- Paramètres : Aucun
- Retour : Aucun

- **Attributs et types**
  - `navigate`: fonction de navigation utilisée pour renvoyer l'utilisateur à la liste des fichiers
  - `location`: contient les informations sur le fichier sélectionné
  - `file`: le fichier sélectionné
  - `fileName`: le nom du fichier sélectionné
  - `fileContents`: le contenu du fichier sélectionné
  - `fileList`: la liste des fichiers parcourus à l'étape précédente
  - `fileThumbnail`: l'image miniature du fichier (première page du fichier)
  - `articleData`: un objet qui contient les articles extraits du fichier
- **Relations d'utilisation**

Les relations d'utilisation du composant `File` avec les autres composants consistent en l'utilisation de la navigation pour renvoyer l'utilisateur à la liste des fichiers s'il souhaite changer de fichier. Ce comportement est réalisé grâce à la fonction `useNavigate`de react-router. Le composant utilise également les attributs de location pour récupérer les informations sur le fichier sélectionné. Ce composant est l'enfant direct du composant `App`.

Composants

Tous les composants écrits dans cette application sont des composants fonctionnels. Je n'ai pas eu besoin d'écrire de composants de classe. Bien que je n'aille pas décrire de nouveau les conteneurs vus à la section précédente, je précise qu'ils sont aussi des composants fonctionnels de l'application.

Card.jsx

Ce composant est une fonction qui définit une carte pour afficher des informations sur un fichier spécifique. Il s'agit d'un composant réutilisable, qui peut être utilisé multiple fois dans l'application avec différentes valeurs de props.

Le composant prend des props en argument, qui sont utilisées pour personnaliser la carte pour le fichier spécifique qu'il représente. Les props comprennent les éléments suivants:

- `name`: le nom du fichier
- `image`: image de la première page du fichier
- `onClick`: une fonction de rappel (callback) qui se déclenche lorsque la carte est cliquée.

Le composant retourne un élément JSX, c'est-à-dire une structure HTML écrite en JavaScript. Le contenu de la carte est défini dans une balise `<li>` qui a une classe CSS `card`. La carte contient également une image et un sous-titre `<h2>` contenant le nom de l'objet.

Lorsque l'utilisateur clique sur la carte, la fonction de rappel `props.onClick()` transmise par le parent `Folder` est appelée.

```
<li className="card" onClick={() => {
      console.log('Clicked on card: ', props.name);
      props.onClick();
}}>
```

Ce composant est utilisé par le composant `Folder`.

About.jsx

`const About` est une fonction qui rend la page “À propos”. Le composant retourne un élément JSX, dans ce cas-ci, une `<div>` contenant d'autres composants: `<Logo />` et `<Navigation />`. Il ne contient pas d’attributs et n’est utilisé que par son composant parent `App`.

Home.jsx

`const Home` est une fonction qui rend la page d'accueil. Le composant retourne un élément JSX, dans ce cas-ci, une `<div>` contenant d'autres composants: `<Logo />`, `<Navigation />` et `<Folder />`. Il ne contient pas d’attributs et n’est utilisé que par son composant parent `App`.

Navigation.jsx

Ce composant est une fonction qui définit un menu de navigation pour l'application. Il utilise le composant `NavLink` fourni par la bibliothèque *React Router* pour définir des liens vers différentes pages de l'application.

Le menu de navigation est défini dans une `<div>` avec une classe CSS `navigation`. À l'intérieur, il y a une liste HTML `<ul>` contenant deux éléments `<li>`, chacun étant un lien vers une page différente.

Les éléments `<li>` sont enveloppés dans des composants `NavLink`, qui sont configurés avec des propriétés spécifiques :

- La propriété `to` indique l'URL de destination du lien.
- La propriété `className` est une fonction qui prend un objet `nav` en argument et retourne une chaîne de caractères représentant les classes CSS à appliquer au lien. Cette fonction est utilisée pour ajouter la classe CSS `nav-active` si le lien est actif (correspond à l'URL actuelle).

Styles

Les styles ont été réalisés en SCSS. SCSS permet d'utiliser des fonctionnalités plus avancées que CSS telles que des variables, des mixins et des imports imbriqués pour faciliter l'écriture et la maintenance du code CSS. Nous avons l’arborescence suivante :

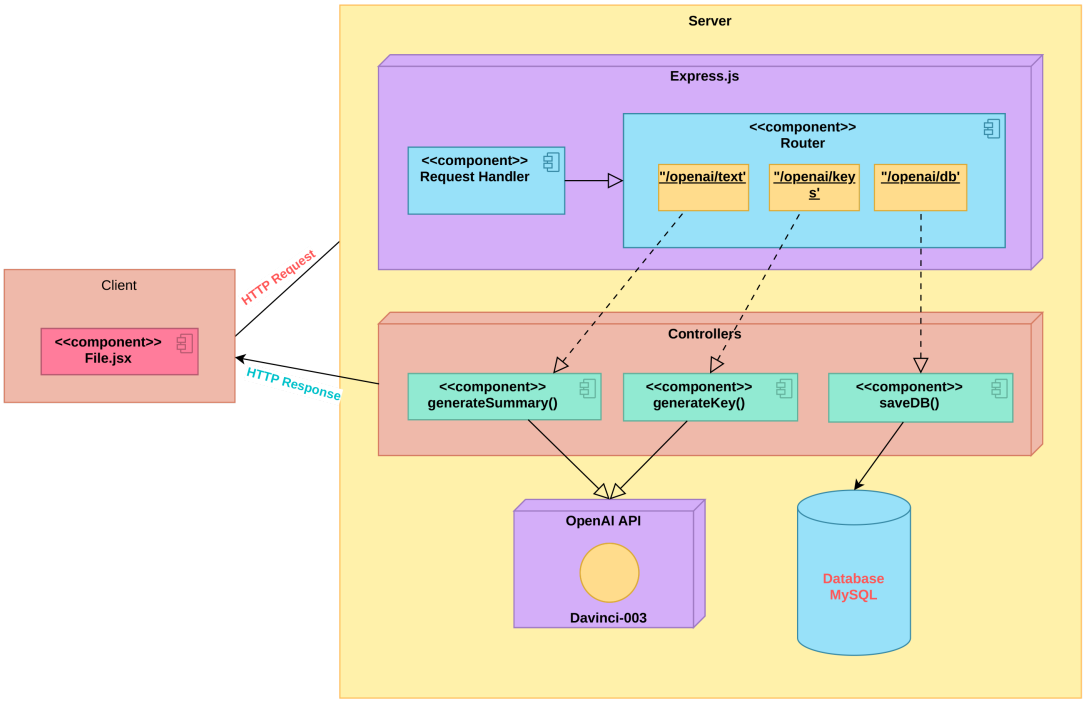
styles/
├── *_settings.scss*
├── *index.scss*

├── ***components/***
    ├── *_card.scss*
    ├── *_navigation.scss*
    ├── *_file.scss*
    ├── *_folder.scss*
    ├── *_logo.scss*

- Le fichier *index.scss* est une feuille de style qui importe différents fichiers SCSS pour construire un site web.
- Le fichier *_settings.scss est la feuille de style qui contient les polices d'écriture, les variables (en l'occurence des couleurs), les mixins et les règles générales de style appliqués à l'ensemble du site (tells que body, *, etc…).*
- Le dossier ***component*** contient les feuilles de style de certains composants React. Chaque composant a sa propre feuille de style qui contient des règles appliquables sur des élèments propres au composant en question.

Backend

Nous avons réalisé le côté serveur avec Node.JS et Express. Node.js est un environnement d'exécution JavaScript côté serveur permettant aussi de créer des serveurs web et Express est un framework web minimaliste pour Node.js. Express fournit une structure de base pour les applications web et facilite la gestion des routes, des middlewares, des requêtes et des réponses HTTP, ainsi que la gestion des sessions et des cookies.



Routes API

/openai/text

URIs

method	endpoint	headers	body	Response	description
`POST`	`/text`	`application/json`	`{articleData : articleData, name : fileName, prompt : prompt, key : apiKey }`	`{ summary: summary }`	Retourne un tableau associatif ayant pour clés les titres des articles et pour valeur les résumés.

/openai/keys

URIs

method	endpoint	headers	body	Response	description
`POST`	`/keys`	`application/json`	`{articleData : articleData, name : fileName, prompt : prompt, key : apiKey }`	`{ keywords: keywords }`	Retourne un tableau associatif ayant pour clés les titres des articles et pour valeur les mots clés.

/openai/db

URIs

method	endpoint	headers	body	Response	description
`POST`	`/db`	`application/json`	`{name: fileName, summary: summary, keywords: keyword}`	`“Tuple created successfully”`	Retourne un message de confirmation.

Contrôleurs et Services

`makeSummary()`

Generates a summary of the file based on user input prompt. Sends a POST request to external server.

- **URL**: `/openai/text`
- **Method**: `POST`
- **Request Body**:

```
{
    articleData: {
        title: string,
        content: string
    },
    name: string,
    prompt: string,
    key: string
}
```

- **Response**:

```
{
    summary: string
}
```

`saveDB()`

Saves the summary and keywords of the file to the database. Sends a POST request to external server.

- **URL**: `/openai/db`
- **Method**: `POST`
- **Request Body**:

```
{
    name: string,
    summary: string,
    keywords: string
}
```

- **Response**:

```
{
    message: string
}
```

`makeIndex()`

Generates keywords of the file based on user input prompt. Sends a POST request to external server.

- **URL**: `/openai/key`
- **Method**: `POST`
- **Request Body**:

```
{
    articleData: {
        title: string,
        content: string
    },
    name: string,
    prompt: string,
    key: string
}
```

- **Response**:

```
{
    keywords: string[]
}
```

Status

Pour la fonction `generateSummary` et `generateKeys` :

- Si la clé API OpenAI est invalide, le code de statut de réponse sera `401 Unauthorized`.
- Si une erreur se produit pendant l'exécution de la fonction, le code de statut de réponse sera `500 Internal Server Error`.
- Si la fonction s'exécute avec succès, le code de statut de réponse sera `200 OK`, et le résumé ou les mots-clés seront renvoyés dans le corps de la réponse.

Pour la fonction `saveDatabase` :

- Si une erreur se produit lors de la connexion à la base de données, le code de statut de réponse sera `500 Internal Server Error`.
- Si un tuple avec le nom donné existe déjà et qu'il est mis à jour avec succès, le code de statut de réponse sera `200 OK`.
- Si un nouveau tuple est créé avec succès, le code de statut de réponse sera `200 OK`.
- Si une erreur se produit pendant l'exécution de la fonction, le code de statut de réponse sera `500 Internal Server Error`.